02

Escapsulamiento

Transcripción

[00:00] Ya vemos cómo poco a poco estamos terminando de construir nuestro sistema bancario. Sabemos cómo referenciar un objeto dentro de otro objeto y sabemos que las reglas del negocio en este caso son que nuestra cuenta va a tener saldo, agencia, número, titular, y fuera de eso va a tener también los métodos depositar, retirar dinero, transferir, digamos que son métodos que hemos creado para nuestras operaciones diarias.

[00:33] Por ejemplo esto me lleva a pensar a mí en un caso de uso bien particular. Aquí, especialmente en retirar, estamos diciendo que si el saldo fuera mayor al valor que hay que retirar, entonces poder retirar, si no retorna un falso. Hasta ahí, todo bien, ya hemos hecho esa prueba, pero no es la única forma en la que podemos modificar el saldo. ¿A qué me estoy refiriendo?

[01:04] Vamos aquí a crear una nueva clase, vamos a llamarla PruebaAcceso. Listo. Ya van a ver por qué le he puesto PruebaAcceso. Quizás se están preguntando por qué pero ya va a llegar ahí la explicación. Creamos nuestro clásico método main y vamos a crear nuestra cuenta. Vamos a ponerle en este caso cuenta "Ctrl + espacio", damos enter, igual new, "Ctrl + espacio", cuenta. Perfecto.

[01:53] Entonces, ya sabemos que a la cuenta le podemos hacer un cuenta.agencia y poner algún valor y eso, y por ejemplo vamos a ponerle a la cuenta que su saldo inicial va a ser igual a 200. Perfecto. Y ahora le vamos a decir cuenta.retirar y supongamos que yo quiero retirar 300, que yo no conozco mi saldo actualmente y que yo quiero retirar 300.

[02:30] ¿Será que yo debería poder tener un saldo negativo en mi cuenta? Va a depender de las reglas de negocio pero lo más probable es que no. Si nosotros ejecutamos ese método, vamos a darle un sysout, imprimimos nuestro saldo, cuenta.saldo, guardamos, ejecutamos nuestro método y vemos que nuestro saldo sigue siendo 200. ¿Por qué?

[03:02] Porque en el caso de retirar, apretamos control y vamos al método directamente, en el caso de retirar él está haciendo la validación, como ya les mencioné antes, para saber si yo puedo o no puedo retirar. ¿Esto en qué me protege? Me protege a que no puedas trabajar directamente con la variable sino simplemente retirar a través de este método y así yo evito tener un saldo negativo.

[03:27] Esto es muy bueno para una cuenta bancaria, pero ¿qué pasaría si yo hago esto de acá? Cuenta.saldo va a ser igual al saldo de mi cuenta menos 300, por ejemplo. Vemos que él está aceptando aquí, le damos guardar, ejecutamos y vemos que tengo -100. Esto no es bueno para el negocio, obviamente, y no está cumpliendo con nuestra regla básica que hemos puesto que es no tener negativos.

[04:09] ¿Cómo podríamos evitar que alguien haga esto en el futuro? Por ejemplo, tú que estás programando en este momento, tú ya sabes que no puedes tener saldo negativo, por lo tanto tú vas a usar el método retirar en caso quieras descontar saldo de esta cuenta. Pero alguien que recién está viniendo digamos a la empresa y está comenzando a ver el código, él no va a tener esta información del todo clara.

[04:37] No sé, quizás podemos escribir las reglas del negocio en algún documento de Word, grabar en un Google Drive, comentarlo aquí en el código, pero de todas formas seguimos siendo muy susceptibles al error humano, de todas formas seguimos desprotegidos ante este caso de uso.

[04:58] En el caso de Java existe un concepto muy fuerte que se llama encapsulamiento. ¿Qué es el encapsulamiento? Vamos a ponerlo con un ejemplo. En el caso de un auto, vamos a suponer el carro, tú manejas el carro, tú sabes conducir, agarras el timón, palanca de cambio pedales, manejas. Tú no intervienes directamente con el motor.

[05:25] El motor está completamente aislado, está encapsulado, tú sabes que el motor funciona, sabes que el motor cumple su trabajo, pero tú no intervienes directamente con él, a no ser que algo esté fallando ahí pero es un caso aislado. Pero en el uso normal tú no intervienes directamente con el motor. Tú intervienes con las herramientas que contactan al motor y el motor ejecuta el trabajo. Eso es encapsulamiento.

[05:50] ¿En qué ayuda? Tú puedes cambiar el motor si quieres, y la experiencia de conducción va a seguir siendo igual, tú igual vas a seguir siendo capaz de conducir el auto así el motor sea primero un motor a gasolina, un motor a gas o un motor a hidrógeno, no lo sé, pero básicamente está encapsulado y aislado. Funciona de una manera no automática pero sí autónoma.

[06:18] Entonces, aplicando eso a este caso de uso en Java tenemos lo que se llaman modificadores de acceso. ¿Ustedes recuerdan aquí esta palabra llamada public, public de aquí? Es hora ya de explicar para qué sirven. Estos son los llamados modificadores de acceso. ¿Qué quiere decir? Que este método de aquí, public, hace que este método sea visible desde cualquier parte de mi código.

[06:49] Y tenemos otra palabra reservada, otro modificador de acceso llamado private, que es ese de aquí. Con private nosotros le decimos: "Esta variable de aquí, el saldo escóndelo, no quiero que nadie trabaje directamente con mi saldo. Nadie puede modificar directamente el saldo". Entonces, si vamos aquí, vamos a ver cómo saldo ya está subrayado en rojo, no está compilando el código, porque como el mensaje dice, la variable cuenta.saldo no es visible.

[07:36] Como está marcada como private, él no es accesible desde otra parte del código que no sea la misma clase con la que está trabajando. Por lo tanto, si yo deseo ya modificar el saldo directamente, va a tener que ser a través de ciertos métodos.